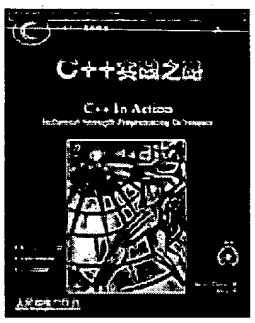
Contents

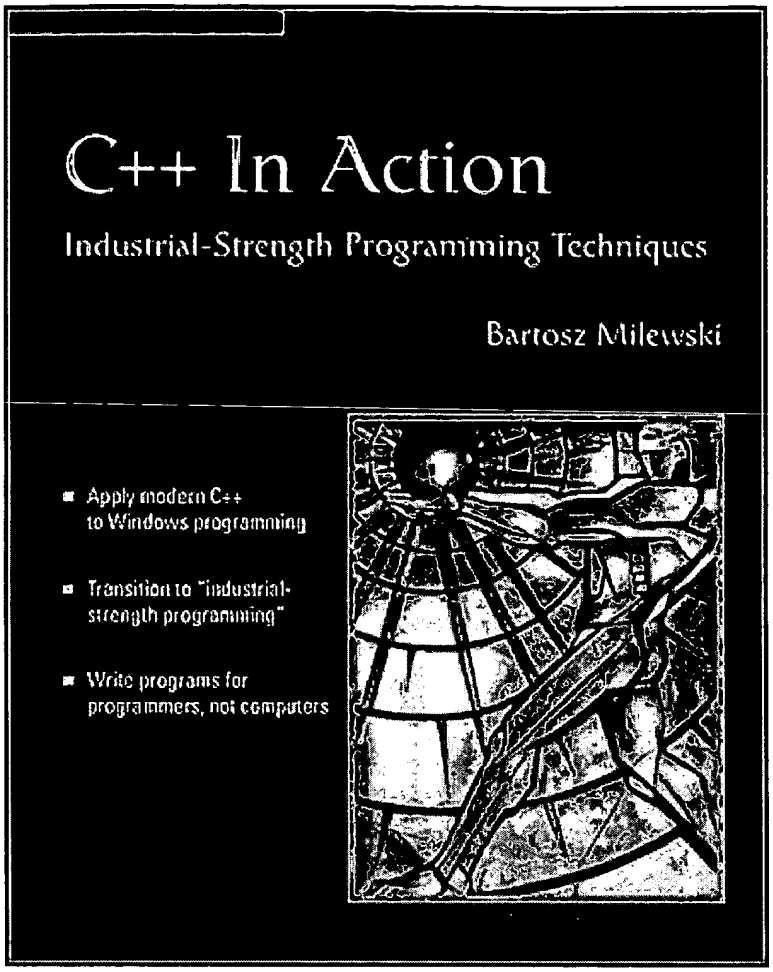


Chinese Edition

The Chinese Translation has been made available in mainland China by Post & Telecom Press. China-Pub.

Questions and Errors

If you find any **errors** in the printed version that are not already listed in the errata, post it on our forum. You can also ask C++ and Windows programming questions there.



Incredible Bargains

You can buy a used copy of the book for **ridiculously low price** (less than $10) through some discount sellers on Amazon. Apparently my book was selected as a bait to attract buyers who might also buy more expensive books. I have no problem with it, as long as it makes the book affordable to more people. So enjoy! You can also buy it new at Barnes and Noble.

Instructions for browsing off-line

Two Years After

I had no idea that "C++ In Action" would become such a cult book. But apparently that's what happened. The book didn't make much money, but it stirred some emotions. People either love it or hate it--sometimes for the same reasons.

The book doesn't lead the reader by the hand in easy little steps. It's not one of the books for dummies, and it doesn't promise to teach you C++ in seven days. To make things worse, it starts at an elementary level and then quickly proceeds to explaining architectural choices. One reader's initial reaction was:

Get this b k NOW..!!!! AWESOME

only to be followed by:

Dissatisfied after getting t the 4th chapter
My initial reaction was to praise the approach taken by the author after reading the

first chapter, however, I became gradually disappointed with the simplistic way the author approached knowledge transfer and therefore had to review my previous rating of this book.

Guilty as charged. I was too greedy, trying to cram years of experience into one volume. Or, you might say, I was just honest. I don't believe in recipes. I don't think you can *program* a programmer. All I wanted to say in this book was, "This is the *kinda* way you should program," with the stress on *kinda*. It is, in a sense, a book about "programming with your right brain," hence I wasn't afraid of inserting some seemingly *paradoxical* statements, like this one (often quoted),

Programming is for programmers, not for computers

There are people who read this sentence and say, "This makes no sense!" and there are people who say, "Finally, somebody said that!"

One of such quotes ended up as a motto for a German article on abstract speech:

Abstrakte Sprachen

"The fact that the program works has no relevance. "
Bartosz Milewski

What I didn't expect is to be appreciated by the Linux crowd. Yet I somehow ended up as a character in a Linux Science Fiction Story in French. Let me quote from *L'Histoire des Pingouins*:

J'ai rencontré un homme étrange dans l'Ether, Sefiroth : il se nomme Milewski Bartosz.

Now, if this doesn't give my book a cult status, I don't know what does...

Preface to the Web edition

This is a ***Web Book***. It is being published on the Internet by me, the author. I realize that the technology is still not there--the technology that would make a web book as easy to read as its paper based relative. Computer monitors still lag behind paper in terms of resolution, portability and price. Fortunately, you won't need to spend hours connected to the Internet--there are easy ways browse the web off-line (see link above).

Other than that, a web book is a great thing for both the author and the reader (or, should I say, the user). For the author it offers a much richer medium. I can control the layout, the fonts, the colors and the pictures. Notice that most paper books are monochromatic--the few that offer color (especially throughout) are quite expensive. On the web, I can go crazy with colors, textures, fonts and pictures. Not that I'm planning to, mind you; I will try to strike a reasonable balance between a typerwriter copy and a tabloid.

But that's only the beginning. The web lets me do things that are quite impossible to do on paper. First of all, a web book is a hypertext document. I can embed jumps to related topics, embed links to in-depth explanations or author's own views--I can digress as much as I want. Second, I can let the reader download code samples and work with them off line. And third, I can add movement and sound to my book and I can make it interactive by embedding Java applets.

The hope is that all these enhancements will not only make the web book more interesting for the reader, but that they will help the learning process. More importantly, though, *Industrial Strength Programming* is part of a bigger, ambitious plan to transition programming into the 21st century. Since form should follow function, what's a better form for 21st century ideas than a web book?

From the financial point of view, a web book brings no direct revenue to the author, although it might increase trafic to the Reliable Software home page and help sell its products. But that's a blessing in disguise, because, having no marketing constraints, I can start publishing this book right now, as-is, and do it in increments (although the book is 90%

written, it will take time to convert it to *meaningful* hypertext). Moreover, I can keep improving the book interactively--something a paper-bound author can't do without waiting for the next edition. It also means that I can accept feedback from readers and respond to legitimate criticism by simply fixing the book.

Compilers

Some code samples on the book's CD-ROM will *not* compile using the (non-standard-compliant) Microsoft Visual C++ v. 6.0. They will compile with the current v. 7.0, a.k.a. **VisualStudio.NET**.
Except maybe for Window-specific programs in the second part of the book, you may try using other compilers. These are some recommendations from the readers (I haven't tried them):

- The Windows based MinGW on http://www.mingw.org/ which is based upon GCC, but do not have any (GNU) license issues.
- The Dev-C++ editor at http://www.bloodshed.net/devcpp.html which is an full-featured Integrated Development Environment (IDE) for Win32 and Linux. It uses GCC, Mingw or Cygwin as compiler and libraries.

Organization of this book

Code samples from this book are directly downloadable in zipped form. Every time you see an icon like the one on the left, you can click on it and download the source.

The image of a rotten apple will accompany sermons about bad programming practices and tips on how to avoid bugs.

Bartosz Milewski, *Aug 1, 1997*

Implementation Digression

Obviously, for polymorphism to work, the object itself must store some information about its actual type. Based on this type the call is dispatched to the appropriate implementation of the method. In fact, the object may store a **pointer** to a dispatch table through which virtual methods are called. And that's how it works.

Every **class** that has at least one virtual method (a polymorphic **class**) has a hidden member that is a **pointer** to a ***virtual table***. The **virtual table** contains the addresses of the virtual member functions for that particular **class**. When a call is made through a **pointer** or reference to such object, the compiler generates code that dereferences the **pointer** to the object's **virtual table** and makes an ***indirect call*** using the address of a member function stored in the table.

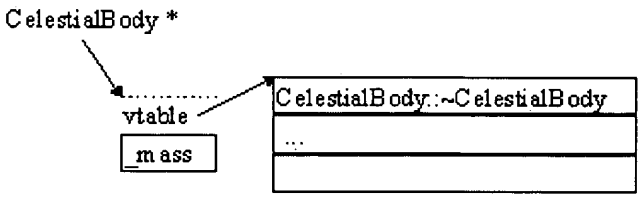


Figure. The invisible member of a celestial body—the **pointer** to the vtable. The first entry in the vtable is a **pointer** to a function that is the destructor of celestial body.

An object of a derived **class** that has overridden the implementation of the virtual method has its **pointer** pointed to a different **virtual table**. In that table the entry corresponding to that particular method contains the address of the overridden member function.

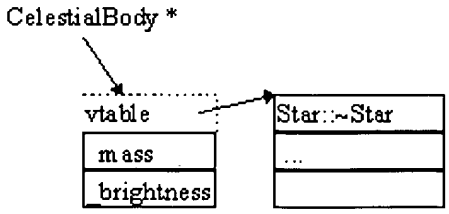


Figure 2-2 The vtable **pointer** of a star points to a different vtable. Its first entry is also a **pointer** to a function that serves as a destructor. Only this time it's the destructor of a star.

Notice that, based on the type of the **pointer**, the compiler has enough information to decide whether to generate an indirect call through a vtable, or a direct call or inline expansion. If the **pointer** is declared to point to a **class** that has virtual functions, and the call is made to one of these functions, an indirect call is automatically generated. Otherwise a direct call or inline expansion is done. In any case, the compiler must first see the declaration of the base **class** (presumably in one of the header files) to know which functions are virtual and which are not.

The Overhead

Before you decide to use virtual functions everywhere without much thinking, I have to

make you aware of the size and speed considerations. The space overhead of polymorphism is: one **pointer** per every instance of the **class**. There is an additional per-**class** overhead of a vtable, but it's not that important. One more **p inter** per object is not a lot when dealing with a small number of relatively large objects. It becomes significant, however, when dealing with a large number of lightweight objects.

A virtual function call is slower than a direct call and significantly slower than the use of an inline function. Again, you can safely ignore this overhead when calling a heavy-weight member function, But turning an inline function such as `AtEnd ()` into a virtual function may significantly slow down your loops.

So don't even think of the idea of creating the **class** `Object`—the mother of all objects—with a handy virtual destructor (and maybe one more integer field for some kind of a **class** ID [run-time typing!], plus some conditionally compiled debugging devices, etc.). Don't try to make it the root of all classes. In fact, if you hear somebody complaining about how slow C++ is, he or she is probably a victim of this Smalltalk syndrome in C++. Not that Smalltalk is a poor language. When size and speed are of no concern, Smalltalk beats C++ on almost all fronts. It is a truly object oriented language with no shameful heritage of the hackers' C. It unifies built-in types with user defined types much better than C++. It has a single-rooted hierarchy of objects. All methods are virtual (you can even override them at runtime!). Java, on the other hand, tries to strike a balance between "objectivity" and performance. In Java all methods are virtual, except when explicitly declared `final` (and then they cannot be overridden).

If speed and size *are* of concern to you, stick to C++ and use polymorphism wisely. Well designed polymorphic classes will lead to C++ code that is as fast as the equivalent C code (and much better from the maintenance point of view). This is possible when polymorphism is used to reduce a string of conditionals (or a switch statement) to a single virtual function call.

The rule of thumb for those coming from the C background is to be on the lookout for the *switch statements* and complicated *conditionals*. It is natural in C++ to use polymorphism in their place.